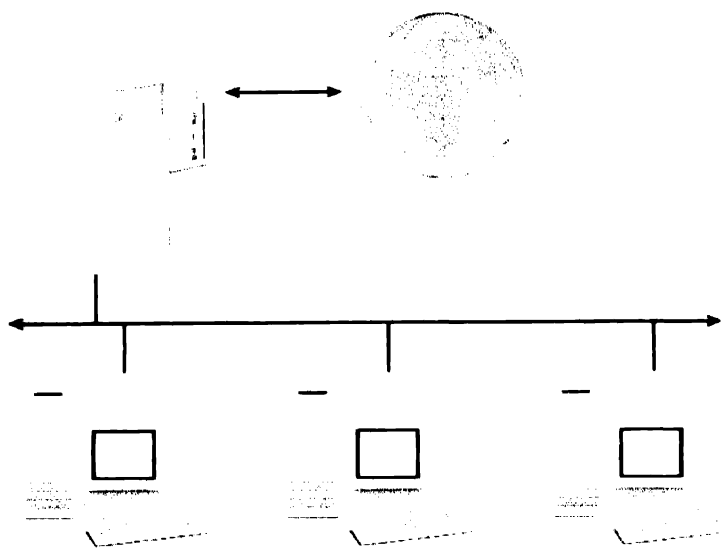
# ICMP
# PING ROUTERS



# CERTIFICATE

This is to certify that the Project entitled, **ICMP PING ROUTERS** is a bonafide work done by **Avdhesh kumar Varun (2K1-COE-17), Dhirender Pratap Singh (2K1-COE-19), Dileep kumar (2K1-COE-20), Vinod kumar Khanghera(2K1-COE-66)** students of VIII SEMESTER (B.E, computer engg.) and it has been carried out under my direct supervision and guidance.

Signature of project guide

Name : **Dr. S.K.SAXENA**

Place : New Delhi

Date : MAY 2005

# ACKNOWLEDGEMENTS

Before we get into project details we would like to add a few heartfelt words for the people who were part of this project in numerous ways... people who gave unending support right from the stage the project idea was conceived. We wish to thank **Dr. S.K.SAXENA** (Project Guide) for helping us developing this project by giving excellent suggestions and making this project worthwhile. We would also like to thank Mr. Sumit kumar, Network engineer (IOCL,Delhi) and Mr. Laxman singh (Software engineer, C-DOT) who keenly inspired and supported us to work on this project.

We also like to express our sense of gratitude to Mr. Manoj Sethi (programmer) and DR. D..R.Choudhary ( H.O.D. computer engg.) for providing information required for the project. finally heartfelt appreciation to our friends who encouraged us from time to time, to an extent that now We firmly believe that this project is an innovative achievement.

**Avdhesh kumar varun** **(2K1-COE-17)**

**Dhirender Pratap singh** **(2K1-COE-19)**

**Dileep kumar** **(2K1-COE-20)**

**Vinod kr. khanghera** **(2K1-COE-66)**

# CONTENTS

**UNIT – 5. COMPREHENSIVE LIFE CYCLE**



***ANNEXURES***

# OVERVIEW

The objective of this software is to meet the growing needs of user-friendly software. This software is mainly pinging software which can ping a specified machine with a unique IP address. Pinging machines (client/host) is used to know whether the machine is alive or dead the destination machine can be on the same network or the same LAN or on different network far away WAN. Through this software one can ping the destination machine directly through a command button by providing the IP address of the destination machine.

The inspiration to develop this software was the result of the fact, we came across with, that our college has developed a lot in last few years and has got computers installed in every branch, hostel ,research, academic, controlling authority (FOT-faculty of technology) and all the computers are connected through a specific LAN in future any of the department may like to access the other department's machine for some data or information .Due to this reason one will like to know that the destination machine is dead or alive, or if the network is alive or dead. One more reason for developing the system is, that the existing system has its own limitations, like it can ping a single destination machine and has a old black MSDOS window with a command line window which is not at all user friendly. Its main advantage is that we can extend it by adding new network technologies (wan, lan, vsat).

In order to meet these requirements, many websites of different companies have been referred-to to collect the related matter for different file formats. This matter contains the basic ideas to fulfill these requirements. The companies want to encourage the use of their controls as constituent controls.

This application wizard is not targeted at experienced users, but is ideal for the user who has a little knowledge of ping and has used other ping software before. It lets you organize different IP address into one convenient database or other text files.

Now, there are so many packages available for the development, we have given emphasis on those available packages, which can be helpful for the developer to work easily and fulfill the user requirements. We have chosen ***Visual Basic 6.0*** in this application; cause developer of today's world always want to create effective and robust applications, packed with impressive outlook in minimum time. Windows programming using C and C++ was a pain to the programmer. One had to write pages of code to display a very simple command button on the screen. Visual Basic makes the life of a programmer much easier by giving an integrated development environment (IDE).

We shall feel highly obliged if the users send us their critical opinion of the presentation and coverage of this software and suggestions to improve the next extended version.

# BACKGROUND

Prior to going into major details, we must go through the vitality of ping process. Ping is a tool that helps to verify IP-level connectivity; Path Ping is a tool that detects packet loss over multiple-hop trips. When troubleshooting, the ping command is used to send an ICMP Echo Request to a target host name or IP address. Use ping whenever you want to verify that a host computer can send IP packets to a destination host. You can also use the ping tool to isolate network hardware problems and incompatible configurations.

After downloading these compressed files (as highlighted in above para) we can't open it until these are transformed into its original form i.e. in decompressed form (original size). In order to do so we must have such a sophisticated software, which must be able to decompress the compressed data, or vice-versa and it should be multi-supportive i.e. to support multiple file formats for the purpose of decompression.

Now the question may arise as to why we use multi-supportive file format only instead of single supported file format. The very simple answer is - that the performance of different decompression formats in different areas has their own unique characteristics and merits .

## OBJECTIVE

The main objective of this software application is to get information about the destination machine such as how far it is, how much data bytes can be sent, is the destination machine dead or alive and to ping more then one destination machine in a single click or a key press through a single wizard.

## SCOPE

The intended product provides efficient service for checking connectivity with other machines. In future, by manipulating some minor changes we can merge this utility in some major applications. For example, if there is any kind of software, which takes data or information from a client or host , then by merging this ICMP_ping utility in it we can make data transfer backup software more sophisticated.

Moreover, we may extend this software application by adding or merging new network technology as they come in use.

## PROBLEM DEFINITION

The motive of this software application is to solve the problem that is being faced in existing systems. During the use of some existing software it was noticed that there is no any common interface to ping and trace some machine or machines .

## USER REQUIREMENTS

User of the system are college staff ,company staff ,database administrator and department heads. Assuming they have very less knowledge of using such type of system, because of that the system has well designed well defined user interface.

In an abstract way it is analyzed that there should be a common and multi supportive interface for pinging and tracing a destination machine .

It is analyzed that user always opt for a single software which is multi supportive instead of single supportive..

As far as the 'memory space' utilized by the software is concerned, it is more in case of single supportive , because you have to install more than one software . On the other hand this software application takes less 'memory space' as compared to previous one, because you have to install one and only one software into the memory.

**USER CHARACTERISTICS**

The main users of this system will be college or company staffs, who are somewhat computer literate and have used software like this before such ping and trace provided by the operating system as a freeware such as Windows-XP, Linux, Macintosh.

## GENERAL CONSTRAINTS

The system only runs on Windows 95, Windows 98, Windows NT, Windows 2000, Windows Professional, Windows XP and other Windows based operating system. It cannot run on UNIX, LINUX, MACINTOSH, and other Non-Windows operating system.

It only pings the Machines of Local Area Network (LAN) and Wide Area Network (WAN) which has been defined in the system. In case any other type of netwok

the system will not support and will not take in account , then an error may occur and no output will be generated.

## GENERAL DESCRIPTION OF INPUTS AND OUTPUTS

The system has two inputs and produces a major output. The general description of these inputs and outputs is given before: --

**INPUT_1**: First input contains the type of network in which the destination system is located it may be of Local Area Network (LAN) and Wide Area Network (WAN) .In this, user chooses one of the type from the drop down list.

**INPUT_2**: Second input is a bit similar to the last one; it contains the IP address or the location of the destination machine on the specified type of network provided in the previous input INPUT_1. In this also the user has to choose one IP address or location from the drop down list.

**OUTPUT_1**: Produces the files that contain the information of the ping execution that have been executed during the usages of the software. It gives information about IP address, Data size, trip time, status, system date, system time for that specific IP address.

**OUTPUT_2**: This output gives the trace report of the IP address provided it gives information about the nodes visited description and trip time.

# DESIGN

The design phase focuses on the detailed implementation of the system recommended in the feasibility study. Emphasis is one translating performance specifications into design specifications. The design phase is transition from a user-oriented document (system proposal) to a document oriented to the programmers or data base personnel.

## Logical and Physical Design

Systems design goes through two phases of development: logical and physical design. A data flow diagram shows the logical flow of a system and defines the boundaries of the system. For a candidate system it describes the inputs (source), outputs (destination), databases (data stores), and procedures (data flows) - all in a format that meets the user's requirements. When analysts prepare the logical system design, they specify the user needs at a level of detail that virtually determines the information flow into and out of the system and the required data resources. The design covers the following:

1. Reviews the current physical system - its data flows, file content, volumes, frequencies, etc.
2. Prepares output specifications - that is, determines the format, content and frequency of reports, including terminal specifications and locations.
3. Prepare input specification - format, content, and most of the input functions. This includes determining the flow of the document from the input data source to the actual input location.

4. Prepares edit, security, and control specifications. This includes specifying the rules for edit correction, backup procedures, and the controls that ensure processing and file integrity.
5. Specifies the implementation plan.
6. Prepares a logical design walkthrough of the information flow, output, input, controls and implementation plan.
7. Reviews benefits, costs, target dates and system constraints.

## Architecture

The following figure shows the overall 3-tier architecture of the proposed system. It is a data-flow based methodology, which defines the modules and their relationships to one another called structured chart. It divides a program into small, independent modules and they are arranged in a hierarchy that approximates a model of the proposed system, which is organized in a top-down or bottom-up approach.

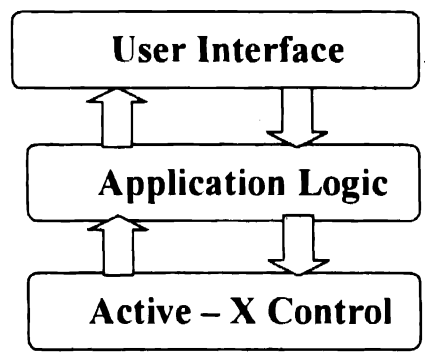



Fig. 3.1 3-tier architecture

Here *User Interface* defines the boundaries of system; *Application Logic* defines the judgment, sense; and reason of system and *Active - X Control* is the basic necessity and emergence of system.

# FUNCTIONAL REQUIREMENTS

The major requirements of this system are the DLL files , which play an important role. Integration of these DLL files generates the desired output. Here following DLLs are used: --

1. Winsock32.DLL
2. ICMP.DLL
3. Kernel32.DLL

All the above-mentioned DLL files should be registered if possible. It contains the desired functions (as described in the *Methodology Used* section), which takes the appropriate input from the end user and generates an output.

# IMPLEMENTATION

The term implementation defines the process of converting a new or revised system design into an operational one. Conversion is one aspect of implementation. The other aspects are the post implementation review and software maintenance.

There are two types of implementation: --

1. Implementation of a new computer system to replace an existing one. This is usually a difficult conversion. If not properly planned, there can be many problems. Some large computer systems have taken as long as a year to convert.

2. Implementation of a modified application to replace an existing one, using the same computer. This type of conversion is relatively easy to handle, provided there are no major changes in the files.

# METHODOLOGY USED

The methodology used in this application software enable the user to make use of those files that are provided by different companies for different formats, so that students can learn from and play with them and company can also get some popularity out of this technology . Basically these files contain the OCX files (which contan the component) and DLL files (run at the time of execution). And may be it also contain some other files that give us the basic hit like how to use these components or related matter.First of all we have to register the ActiveX component (if available) in the component list of development environment.

*How to manually register the ActiveX component:---*

The registration process referred to here is not related to licensing.it refers to the Process of making the control known to the operating system.If you don't use an Install program with this capability built-in , you can register your controls manually.

From the Run option in the Windows 98 or Windows NT start menu,or from a command prompt,run the following command to register an ActiveX control:

*Regsvr32 "path and filename of the ActiveX control's DLL file "*

You will then see a message box that either displays "Load library failed " or Displays "LoadLibrary succeeded ".If you get the success message, then the control

is now registered on the machine and is ready for user by applications or in your Development environment. If you get the failure message, make sure you specified a correct path and filename, and placed it in quotes if it has spaces in it.

***Registration problems:---***

If you are unable to register a fully-self contained ActiveX control on a target Machine,even after following the instructions above , we are currently aware of Only one possible problem that may prevent proper registration of ActiveX Controls..

***If the control is not in the components list,perform the following steps:----***

- Try reinstalling the ActiveX component from its setup program.Continue only Only if that didn't help.
- Click the Browse... button.
- Click on all files in the files of type tab.
- Find and select the component (.DLL) file (use the one in the windows system Folder that was copied there by the setup program) then click on the OK button.
- The component entry should now appear in the components list.

# API function used

- **WSACleanUp**
- **WSAStartUP**
- **Inet_addr**
- **GetHostByName**
- **GetHostByAddr**
- **CopyMemory**
- **LStrlenA**
- **Icmpcreatefile**
- **Icmpclosehandle**
- **Icmpsendecho**

The description of various functions used is given here

- **WSACleanup** (winsock32.dll)

  It has got no arguments.

  Return Values

  Zero indicates success. **SOCKET_ERROR** indicates failure. To get a specific error value, call **WSAGetLastError**. In a multithreaded environment, **WSACleanup** terminates Windows Sockets operations for all threads. This function initiates no action and is provided only for compatibility reasons.

- **WSAStartUp** (winsock32.dll)

  The **WSAStartup** function *must* be the first Windows Sockets function called by an application or DLL. It allows an application or DLL to specify the version of Windows Sockets required and retrieve details of the specific Windows Sockets implementation. The application or DLL can only issue further Windows Sockets functions after successfully calling WSAStartup.

  It has got following arguments:

  - ***WVersionRequired***

    Where WversionRequired is the highest version of Windows Sockets support that the caller can use.

  - ***lpWSAdata***

    Pointer to the WSADATA data structure that is to receive details of the Windows Sockets implementation.

  **Return Values**

  The WSAStartup function returns zero if successful. Otherwise, it returns one of the error codes listed in the following. An application cannot

call WSAGetLastError to determine the error code as is normally done in Windows Sockets if WSAStartup fails

Calling the WSAStartup function initializes the Winsock.dll and a WSADATA structure that contains the details of the Winsock implementation. When an application or DLL has finished using the Winsock.dll, it must call WSACleanup to enable the Winsock.dll to free any resources for the application. For every call to WSAStartup, there must be a call to WSACleanup.

The WSADATA structure pointed to by *lpWSAData* stores Winsock initialization data returned by a call to WSAStartup. WSADATA contains Winsock.dll implementation data. An application or DLL can call WSAStartup repeatedly if it needs to obtain the WSADATA structure data more than once.

- **Inet_addr** (winsock32.dll)

This function converts a string containing a dotted address into a network address in the format of an IN_ADDR structure.This function has only one argument.

- *cp* Null-terminated string that represents a number expressed in the Internet standard "." (Dotted) notation

**Return Values**

An unsigned long value containing a suitable binary representation of the specified Internet address indicates that no error occurred. INADDR_NONE indicates that the string in the *cp* parameter does not contain a legitimate Internet address, for example, if a portion of an "a.b.c.d" address exceeds 255.

- **GetHostByAddr** (winsock32.dll)

The gethostbyaddr function retrieves the host information corresponding to a network address.

This function has got following arguments.

- ***Addr*** Pointer to an address in network byte order.
- ***len*** Length of the address.
- ***type*** Type of the address, such as the AF_INET address family type (defined as TCP, UDP, and other associated Internet protocols). Address family types and their corresponding values are defined in the winsock2.h header file.

**Return Values**

If no error occurs, gethostbyaddr returns a pointer to the HOSTENT structure. Otherwise, it returns a NULL pointer, and a specific error code can be retrieved by calling WSAGetLastError.

**GetHostByName** (winsock32.dll)

The gethostbyname function retrieves host information corresponding to a host name from a host database.

It accepts following arguments

- ***Name*** Pointer to the null-terminated name of the host to resolve.

**Return Values**

If no error occurs, gethostbyname returns a pointer to the hostent structure described above. Otherwise, it returns a NULL pointer and a specific error number can be retrieved by calling WSAGetLastError.

- **CopyMemory** (kernal32.dll)

  The CopyMemory function copies a block of memory from one location to another. It has following arguments:

  - ***Destination*** Pointer to the starting address of the copied block's destination.
  - ***Source*** Pointer to the starting address of the block of memory to copy.
  - ***Length*** Specifies the size, in bytes, of the block of memory to copy.

  Return Values This function has no return value.

  If the source and destination blocks overlap, the results are undefined. For overlapped blocks, use the MoveMemory function. The first parameter, *Destination*, must be large enough to hold *Length* bytes of *Source*; otherwise, a buffer overrun may occur.

- **Lstrlen** (kernel32.dll)

  The lstrlen function returns the length in bytes (ANSI version) or WCHARs (Unicode version) of the specified string (not including the terminating null character). It has got the following arguments:

  - ***lpString*** Pointer to a null-terminated string.

  **Return Values**

  The return value specifies the length of the string, in TCHARs. This refers to bytes for ANSI versions of the function or WCHARs for Unicode versions.

- **IcmpCreateFile** (Icmp.dll)

  This function creates a handle on which Internet Control Message Protocol (ICMP) requests can be issued. This function has got no arguments.

**Return Values**

An ICMP handle indicates success. INVALID_HANDLE_VALUE indicates failure. To get extended error information, call GetLastError.

- **IcmpCloseHandle** (Icmp.dll)

This function closes an Internet Control Message Protocol (ICMP) handle opened by IcmpCreateFile. It has following arguments:

- ***IcmpHandle*** ICMP handle opened by IcmpCreateFile.

Return Values

TRUE indicates success. FALSE indicates failure.

# DATA STRUCTURES USED

Various data structures used in the project are given below:

## WSADATA Structure

The **WSADATA** structure has the following form:

```
Private Type WSADATA
        wVersion As Integer
        wHighVersion As Integer
        szDescription (0 To 255) As Byte
        szSystemStatus (0 To 128) As Byte
        imaxsockets As Integer
        iMaxUdpDg As Integer
        lpVendorInfo As Long
End Type
```

The **WSADATA** structure is used to store Windows Sockets initialization information returned by a call to the **AfxSocketInit** global function.

**Members**

- **WVersion**

  The version of the Windows Sockets specification that the Windows Sockets DLL expects the caller to use.

- **WhighVersion**

  The highest version of the Windows Sockets specification that this DLL can support (also encoded as above). Normally this is the same as **wVersion**.

- **SzDescription**

  A null-terminated ASCII string into which the Windows Sockets DLL copies a description of the Windows Sockets implementation, including vendor identification.

- **SzSystemStatus**

  A null-terminated ASCII string into which the Windows Sockets DLL copies relevant status or configuration information.

- **IMaxSockets**

  The maximum number of sockets which a single process can potentially open. Obviously there is no guarantee that a particular application can actually allocate **iMaxSockets** sockets, since there can be other Windows Sockets applications in use.

- **IMaxUdpDg**

  The size in bytes of the largest User Datagram Protocol (UDP) datagram that can be sent or received by a Windows Sockets application. If the implementation imposes no limit, **iMaxUdpDg** is zero. In many implementations of Berkeley sockets, there is an implicit limit of 8192 bytes on UDP datagrams (which are fragmented if necessary). A Windows Sockets implementation can impose a limit based, for instance, on the allocation of fragment reassembly buffers. The minimum value of **iMaxUdpDg** for a compliant Windows Sockets implementation is 512.

- **LpVendorInfo**

  A far pointer to a vendor-specific data structure. The definition of this structure (if supplied) is beyond the scope of the Windows Sockets specification.

## HOSTENT

The **hostent** structure is used by functions to store information about a given host, such as host name, IP address, and so forth. An application should never attempt to modify this structure or to free any of its components. Furthermore, only one copy of the **hostent** structure is allocated per thread, and an application should therefore copy any information that it needs before issuing any other Windows Sockets API calls. The hostent structure is of the following form:

```
Private Type Hostent
        h_name As Long
        h_aliases As Long
        h_addrtype As Integer
        h_length As Integer
        h_addr_list As Long
End Type
```

Members

- **h_name**

  Official name of the host (PC). If using the DNS or similar resolution system, it is the Fully Qualified Domain Name (FQDN) that caused the server to return a reply. If using a local hosts file, it is the first entry after the IP address.

- **h_aliases**

Null-terminated array of alternate names.

- **h_addrtype**

  Type of address being returned.

- **h_length**

Length of each address, in bytes.

- **h_addr_list**

  Null-terminated list of addresses for the host. Addresses are returned in network byte order. The macro **h_addr** is defined to be **h_addr_list[0]** for compatibility with older software.

## IP_OPTION_INFORMATION

The ip_option_information structure describes the options to be included in the header of an IP packet. The TTL, TOS, and Flags values are carried in specific fields in the header. The OptionsData bytes are carried in the options area following the standard IP header. The IP_OPTION_INFORMATION structure is of the following form:

```
Private Type IP_OPTION_INFORMATION
        ttl As Byte
        Tos As Byte
        Flags As Byte
        OptionsSize As Long
        OptionsData As Long
End Type
```

Members

- **Ttl**

String containing the Time-to-live.

- **Tos**

String containing the type of service

- **Flags**

String containing IP header flags.

- **OptionsSize**

String containing the domain name or the workgroup name.

- **OptionsData**

String containing supported IP option types. These types define the options which may be used in the OptionsData field of the IP_OPTION_INFORMATION structure.

## IP_ECHO_REPLY

The icmp_echo_reply structure describes the data returned in response to an echo request (**IcmpSendEcho**).

The ip_echo_reply structure is of the following form:

*Private Type IP_ECHO_REPLY*

*Address As Long*

**Status As Long**

*RoundTripTime As Long*

***DataSize As Integer***

*Reserved As Integer*

*data As Long*

*Options As IP_OPTION_INFORMATION*

*ReturnedData As String * 256*

*End Type*

**Members**

- ***Address***

Contains the replying IP address

- ***Status***

Contains the reply IP_STATUS.

- ***RoundTripTime***

  RTT in milliseconds.

- ***DataSize***

  Contains the reply data size in bytes.

- ***Reserved***

  Reserved for system use.

- ***Data***

  Contains a pointer to the reply data.

- ***Options***

  String containing reply

# SYSTEM REQUIREMENT SPECIFICATIONS

## Minimum Requirement

| Processor Required | Memory Required | Operating System |
|---|---|---|
| Pentium-II (minimum) | 64 Mb Ram,<br>100 MB Hard Disk | Windows 95 |

## Recommended Requirement

| Processor Required | Memory Required | Operating System |
|---|---|---|
| Pentium-III, IV | 128 Mb Ram,<br>100 MB Hard Disk | Windows 98, 2000,XP |

# ABOUT VISUAL BASIC

Welcome to Microsoft Visual Basic, the fastest and easiest way to create applications for Windows. Whether you are an experienced professional or brand new to Windows programming, Visual Basic provides you with a complete set of tools to simplify rapid application development.

**So what is Visual Basic?** The "**Visual**" part refers to the method used to create the graphical user interface (GUI). Rather than writing numerous lines of code to describe the appearance and location of interface elements, you simply add pre-built objects into place on screen. If you've ever used a drawing program such as Paint, you already have most of the skills necessary to create an effective user interface.

The "**Basic**" part refers to the BASIC (Beginners All-Purpose Symbolic Instruction Code) language, a language used by more programmers than any other language in the history of computing. Visual Basic has evolved from the original BASIC language and now contains several hundred statements, functions, and keywords, many of which relate directly to the Windows GUI. Beginners can create useful applications by learning just a few of the keywords, yet the power of the language allows professionals to accomplish anything that can be accomplished using any other Windows programming language.

The Visual Basic programming language is not unique to Visual Basic. The Visual Basic programming system, Applications Edition included in Microsoft Excel, Microsoft Access, and many other Windows applications uses the same language. The Visual Basic Scripting Edition (VBScript) is a widely used scripting language and a subset of the Visual Basic language. The investment you make in learning Visual Basic will carry over to these other areas.

Whether your goal is to create a small utility for yourself or your work group, a large enterprise-wide system, or even distributed applications spanning the globe via the Internet, Visual Basic has the tools you need.

- Data access features allow you to create databases, front-end applications, and scalable server-side components for most popular database formats, including Microsoft SQL Server and other enterprise-level databases.
- ActiveX technologies allow you to use the functionality provided by other applications, such as Microsoft Word word processor, Microsoft Excel spreadsheet, and other Windows applications. You can even automate applications and objects created using the Professional or Enterprise editions of Visual Basic.
- Internet capabilities make it easy to provide access to documents and applications across the Internet or intranet from within your application, or to create Internet server applications.
- Your finished application is a true .exe file that uses a Visual Basic Virtual Machine that you can freely distribute.

**The Structure of Visual Basic Application**

An application is really nothing more than a set of instructions directing the computer to perform a task or tasks. The structure of an application is the way in which the instructions are organized; that is, where the instructions are stored and the order in which instructions are executed.

Simple applications such as the classic "hello world" example have a simple structure; organization isn't very important with a single line of code. As

applications become more complex, the need for organization or structure becomes obvious. Imagine the chaos that would result if your application's code were allowed to execute in random order. In addition to controlling the execution of an application, the structure is important to the programmer: how easily can you find specific instructions within your application?

Because a Visual Basic application is based on objects, the structure of its code closely models its physical representation on screen. By definition, objects contain data and code. The form that you see on screen is a representation of the properties that define its appearance and intrinsic behavior. For each form in an application, there is a related *form module* (with file name extension .frm) that contains its code.

**Form and its related form module**

Each form module contains *event procedures* - sections of code where you place the instructions that will execute in response to specific events. Forms can contain controls. For each control on a form, there is a corresponding set of event procedures in the form module. In addition to event procedures, form modules can contain general procedures that are executed in response to a call from any event procedure.

Code that isn't related to a specific form or control can be placed in a different type of module, a *standard module* (.BAS). A procedure that might be used in response to events in several different objects should be placed in a standard module, rather than duplicating the code in the event procedures for each object.

A *class module* (.CLS) is used to create objects that can be called from procedures within your application. Whereas a standard module contains only code, a class module contains both code and data - you can think of it as a control without a physical representation.

While "Managing Projects" describes which components you can add to an application, this chapter explains how to write code in the various components that make up an application. By default, your project contains a single form module. You can add additional form, class, and standard modules, as needed. Class modules are discussed in "Programming with Objects."

**The Structure of a Visual Basic Project**

The following sections describe the different types of files and objects that you can include in a project.

**Form Modules**

Form modules (.frm file name extension) can contain textual descriptions of the form and its controls, including their property settings. They can also contain form-level declarations of constants, variables, and external procedures; event procedures; and general procedures.

**Class Modules**

Class modules (.cls file name extension) are similar to form modules, except that they have no visible user interface. You can use class modules to create your own objects, including code for methods and properties.

**Standard Module**

Standard modules (.bas file name extension) can contain public or module-level declarations of types, constants, variables, external procedures, and public procedures.

**Resource Files**

Resource files (.res file name extension) contain bitmaps, text strings, and other data that you can change without having to re-edit your code. For example, if you plan to localize your application in a foreign language, you can keep all of the user-interface text strings and bitmaps in a resource file, which you can then localize instead of the entire application. A project can contain no more than one resource file.

**ActiveX documents**

ActiveX documents (.dob) are similar to forms, but are displayable in an Internet browser such as Internet Explorer. The Professional and Enterprise editions of Visual Basic are capable of creating ActiveX documents.

**User Control and Property Page Modules**

User Control (.ctl) and Property Page (.pag) modules are also similar to forms, but are used to create ActiveX controls and their associated property pages for displaying design-time properties. The Professional and Enterprise editions of Visual Basic are capable of creating ActiveX controls.

**Components**

In addition to files and modules, several other types of components can be added to the project.

**ActiveX Controls**

ActiveX controls (.ocx file name extension) are optional controls, which can be added to the toolbox and used on forms. When you install Visual Basic, the files containing the controls included with Visual Basic are copied to a common directory (the \Windows\System subdirectory under Windows 95). Additional

ActiveX controls are available from a wide variety of sources. You can also create your own controls using the Professional or Enterprise editions of Visual Basic.

**Insertable Objects**

*Insertable objects*, such as a Microsoft Excel Worksheet object, are components you can use as building blocks to build integrated solutions. An *integrated solution* can contain data in different formats, such as spreadsheets, bitmaps, and text, which were all created by different applications.

**References**

You can also add references to external ActiveX components that may be used by your application. You assign references by using the References dialog, accessed from the References menu item on the Project menu.

**ActiveX Designers**

ActiveX designers are tools for designing classes from which objects can be created. The design interface for forms is the default designer. Additional designers may be available from other sources.

**Standard Controls**

Standard controls are supplied by Visual Basic. Standard controls, such as the command button or frame control, are always included in the toolbox, unlike ActiveX controls and insertable objects, which can be removed from or added to the toolbox.

**Forms, Controls, and Menus**

The first step to creating an application with Visual Basic is to create the interface, the visual part of the application with which the user will interact. Forms and

controls are the basic building blocks used to create the interface; they are the objects that you will work with to build your application.

Forms are objects that expose properties, which define their appearance, methods, which define their behavior, and events, which define their interaction with the user. By setting the properties of the form and writing Visual Basic code to respond to its events, you customize the object to meet the requirements of your application.

Controls are objects that are contained within form objects. Each type of control has its own set of properties, methods and events that make it suitable for a particular purpose. Some of the controls you can use in your applications are best suited for entering or displaying text. Other controls let you access other applications and process data as if the remote application was part of your code.

## ER & DFD'S

The flowchart of the proposed system is given below -

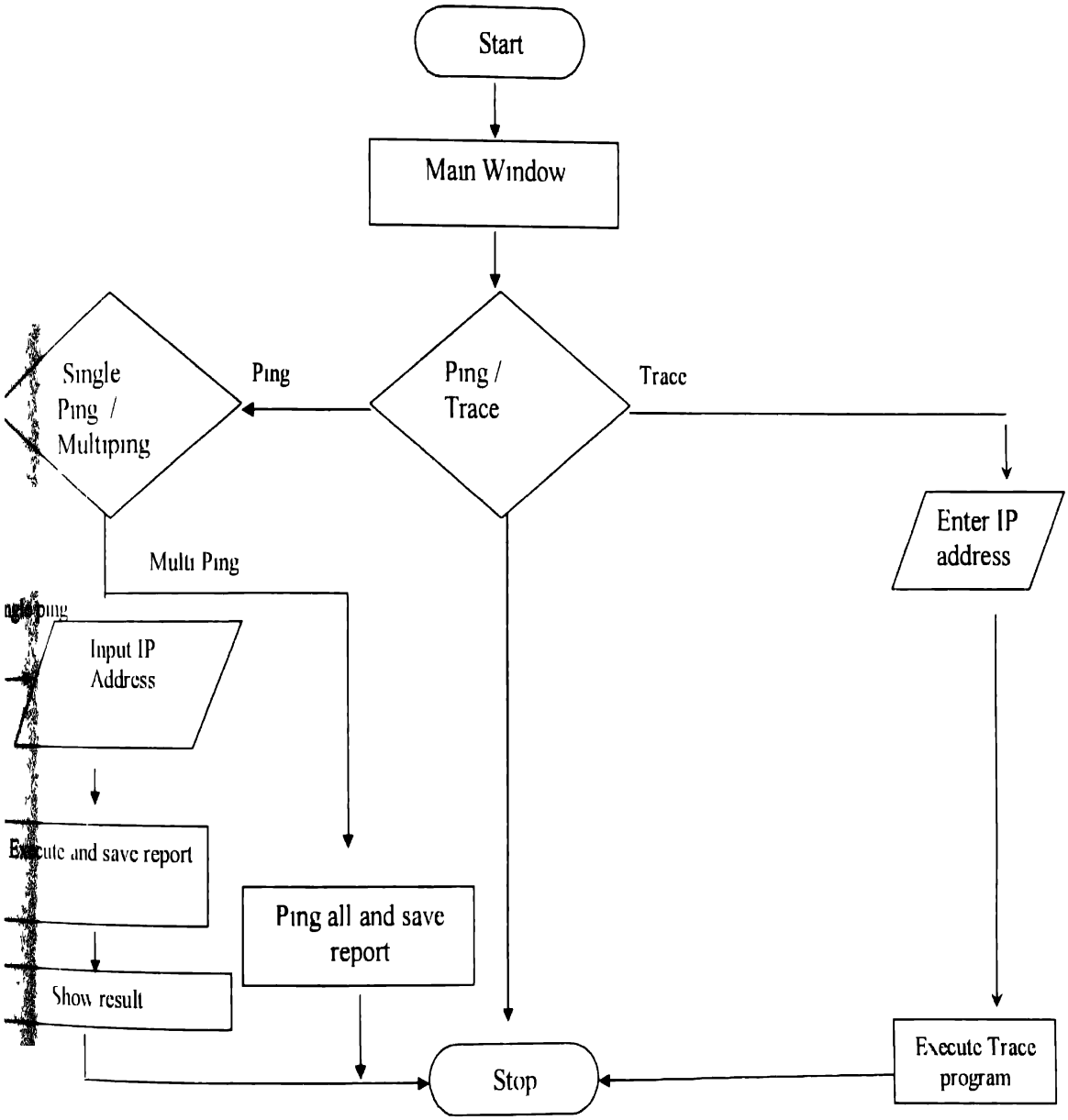
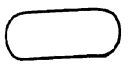


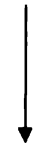

Fig 5 2

In the above fig. Following symbols are used:



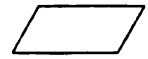
= begin or terminate.

= process



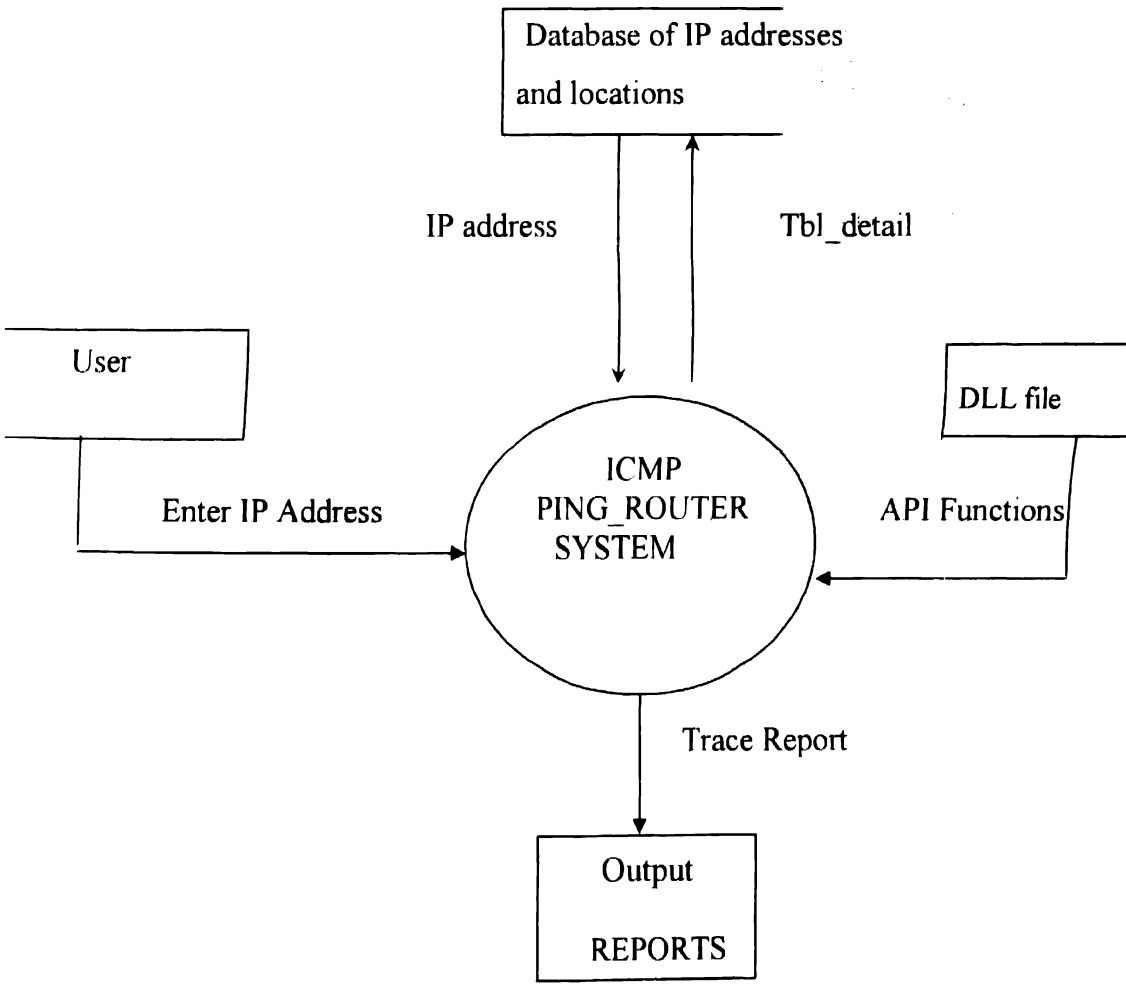
= flow

=decision



=input

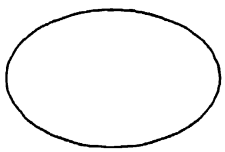
The graphic representation of data movement, process involved and files (data store) used in support of *file decompression system* is given below:--



Here, in the above figure, user browses the required file, which he wants to compress or decompress, and the system take that file from the location prescribed by the user. Then system applies the appropriate function, which has taken from the DLL file and send the output (i.e. in compressed or decompressed form) to the specified location (by user).

In the above figure following symbols are used

= Source or destination



= Process transforming data flow

= Data store



= Data store

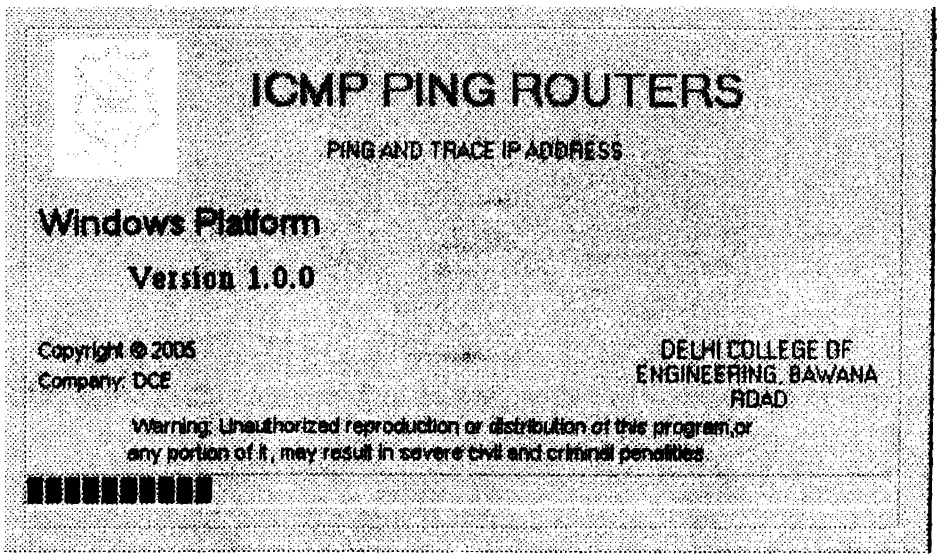
# INPUT AND OUTPUT SCREENS

A common windows user can easily handle this interface; the features of this interface are that, that it is a mouse-enabled system i.e. one can easily handle all facilities by using scrolling mouse.

Another feature is related to GUI i.e. it is a graphic user interface, which contains all preferable tools in windows, like menus , buttons, textboxes, lists, and other related graphic options.

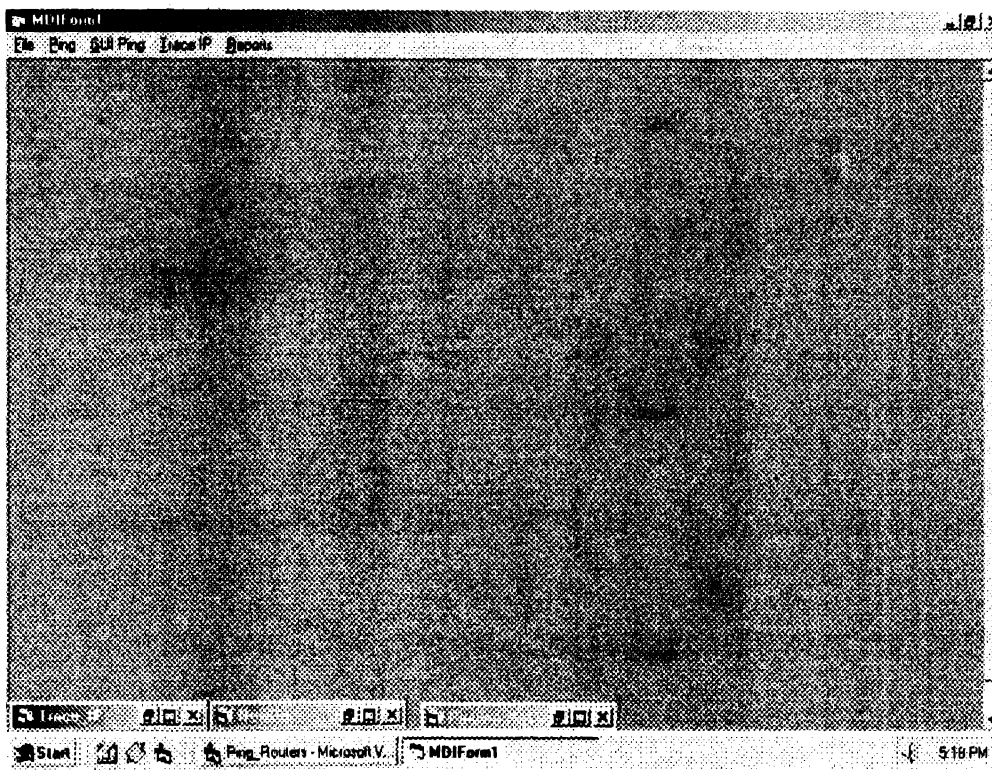
## SOFTWARE INTERFACE

1. Loading Form(frm_splash): --



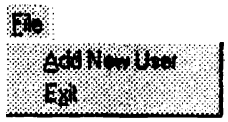
It loads all the necessary settings on your system, within the processing time. It also shows all the labels of application, like name of application, its version, and license to, platform dependent,Company name,copyright,warning and a processing bar(i.e time taken to loading all proper settings).

## 2. Main Menu Form(mdiform): --




**The main menu form contains five menus, which are described below: --**

(i) File Menu: --




In this menu there are two options with shortcut keys.

- **Add new User** (ctrl+a):--If you want to add a new user , then this option will be in use. Its shortcut key is Ctrl+A (i.e. rather than clicking on this option you can press this shortcut key on keyboard).
- **EXIT** (Ctrl+E): -- this option can be used to exit from the system.

(ii) Ping Menu: --


In this menu option there are two options: --

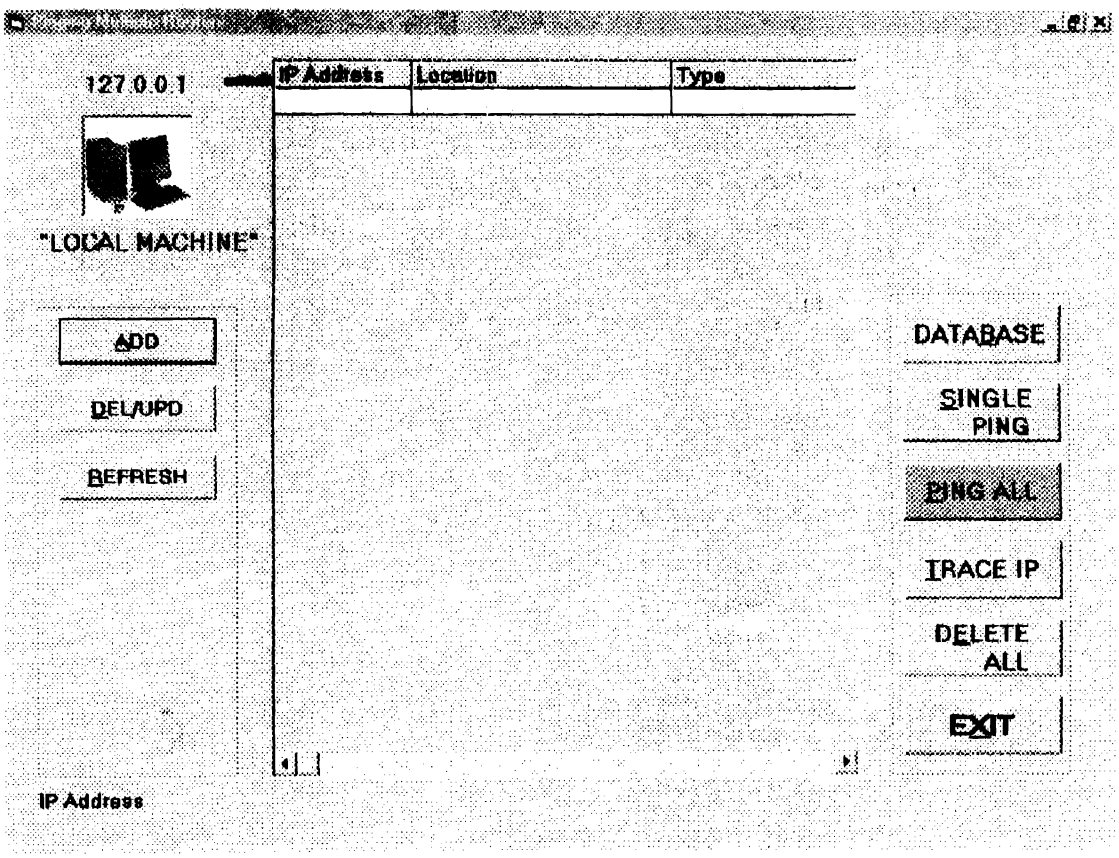
- **Single ping**: -- This option can be used to ping a single destination machine.
- **Multi Ping**:-- This option can be used to ping a no of routers all at once.

(iii) **GUI ping** : this option can be used to ping a no of known routers at a single click and display gui interface.

(iv) **Trace** : this option can be used to trace the ip address of the destination.

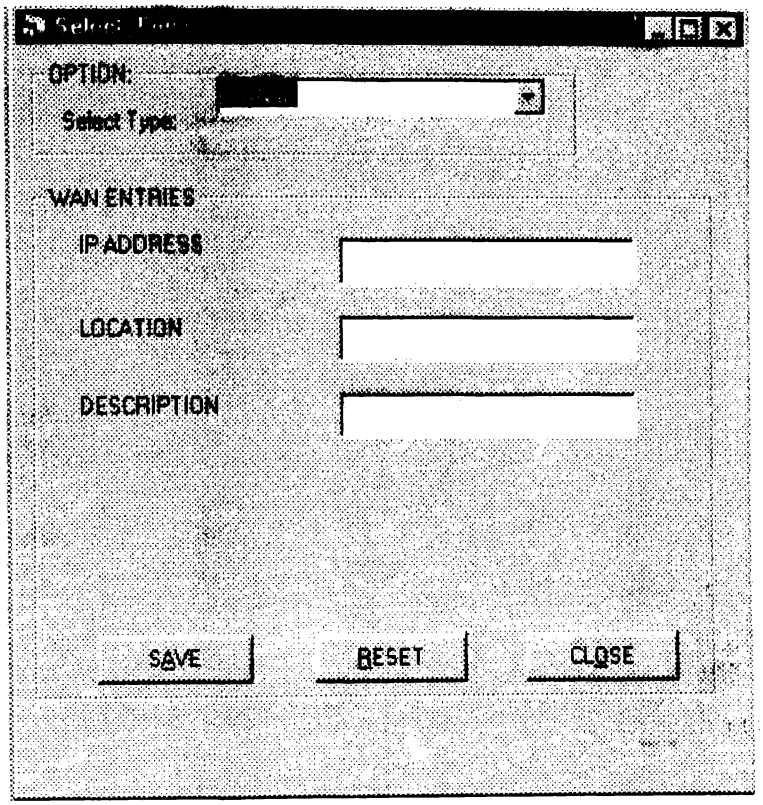
(v) **Reports**: this option is used to display the reports genetrated by the execution of ping program from the database.

**3. Multiping:** --If user clicks on ping->multiping , then this form is displayed.

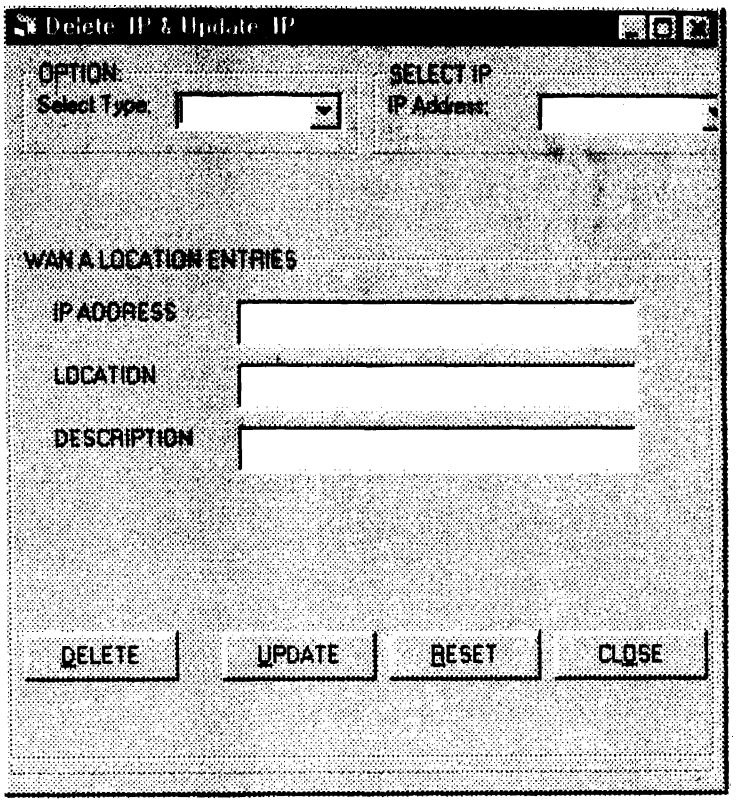



It holds the following facilities: --

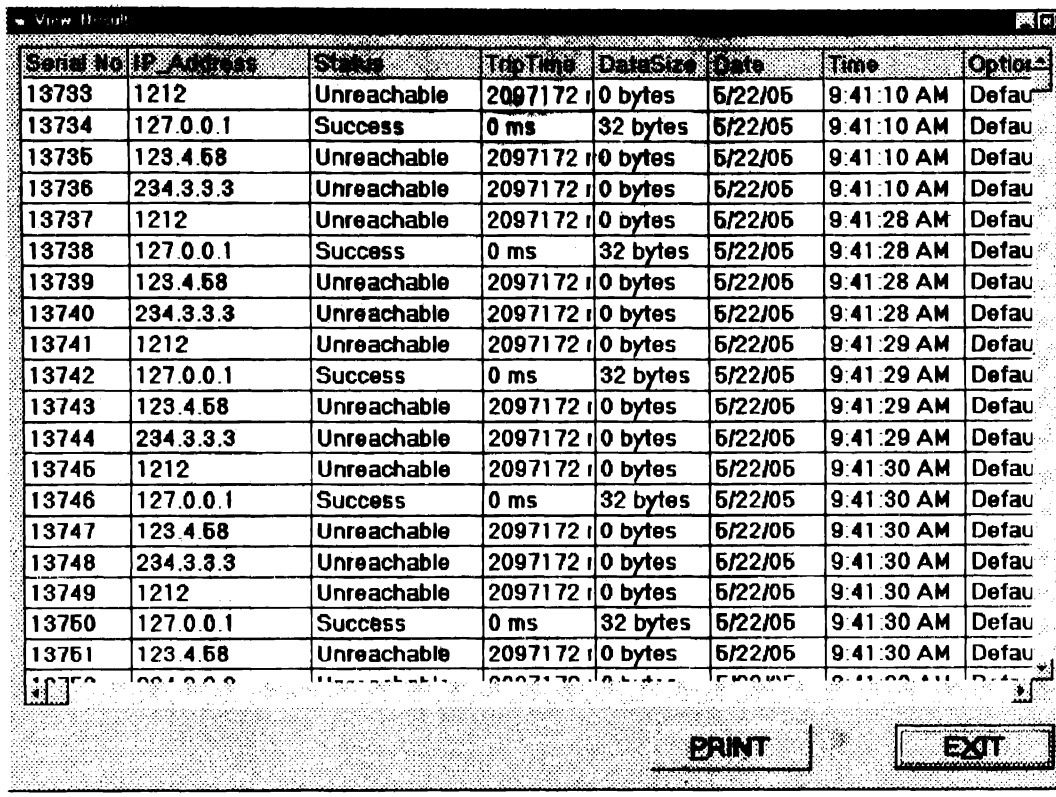
- Add : this option can be used to add an IP addresss of any computer ,may be LAN or WAN




- DEL/UPD : this option is used to delete or update the existing data from the database..



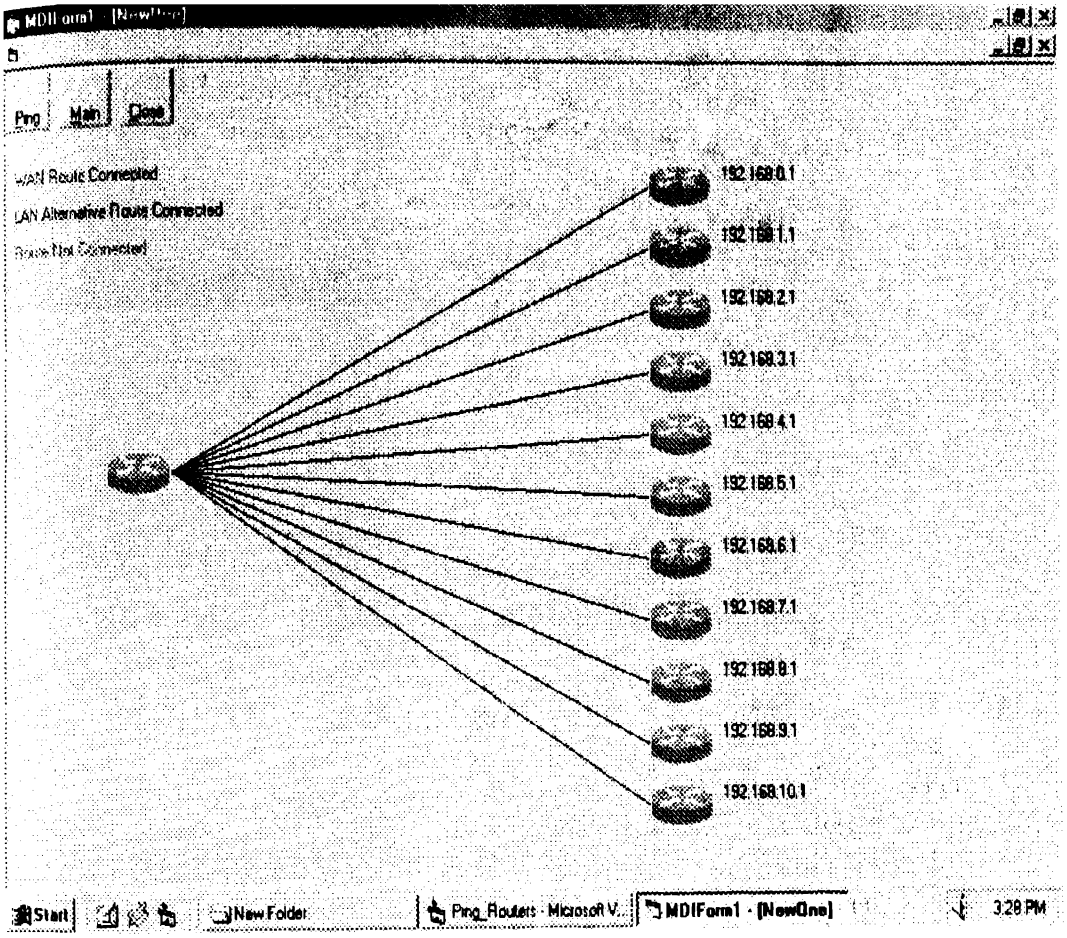

- **Database:** this option displays the whole database of all executed pings and the related information.



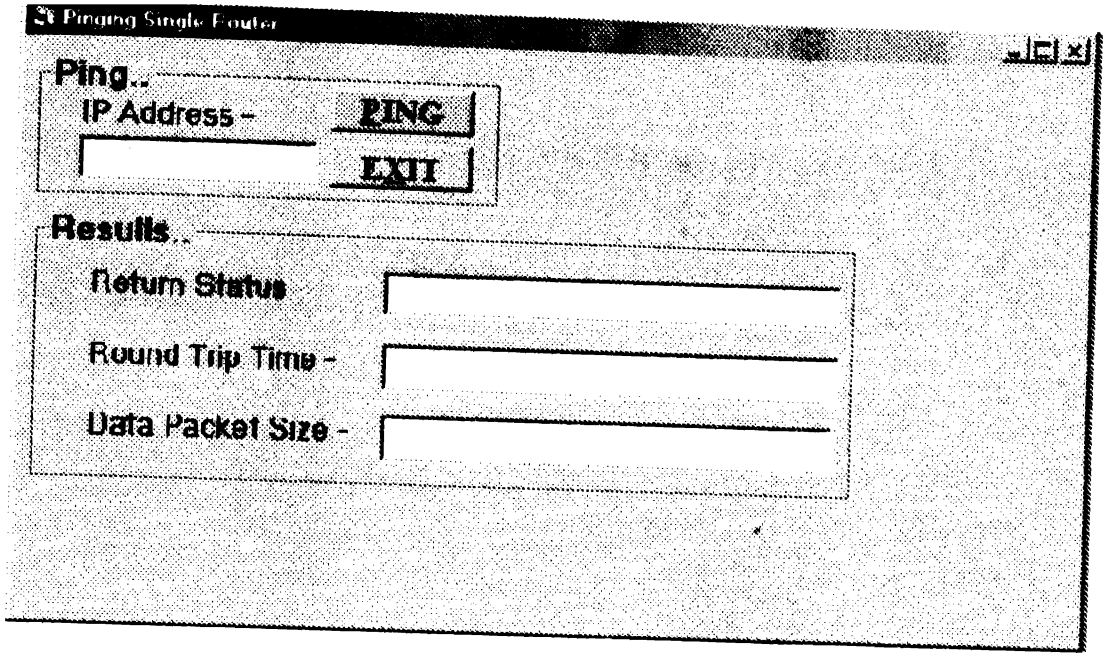

- **Single ping**: this option is used to ping the selected single record from the database .

- **Ping all :** this option is used to ping all the addresses present in the database.

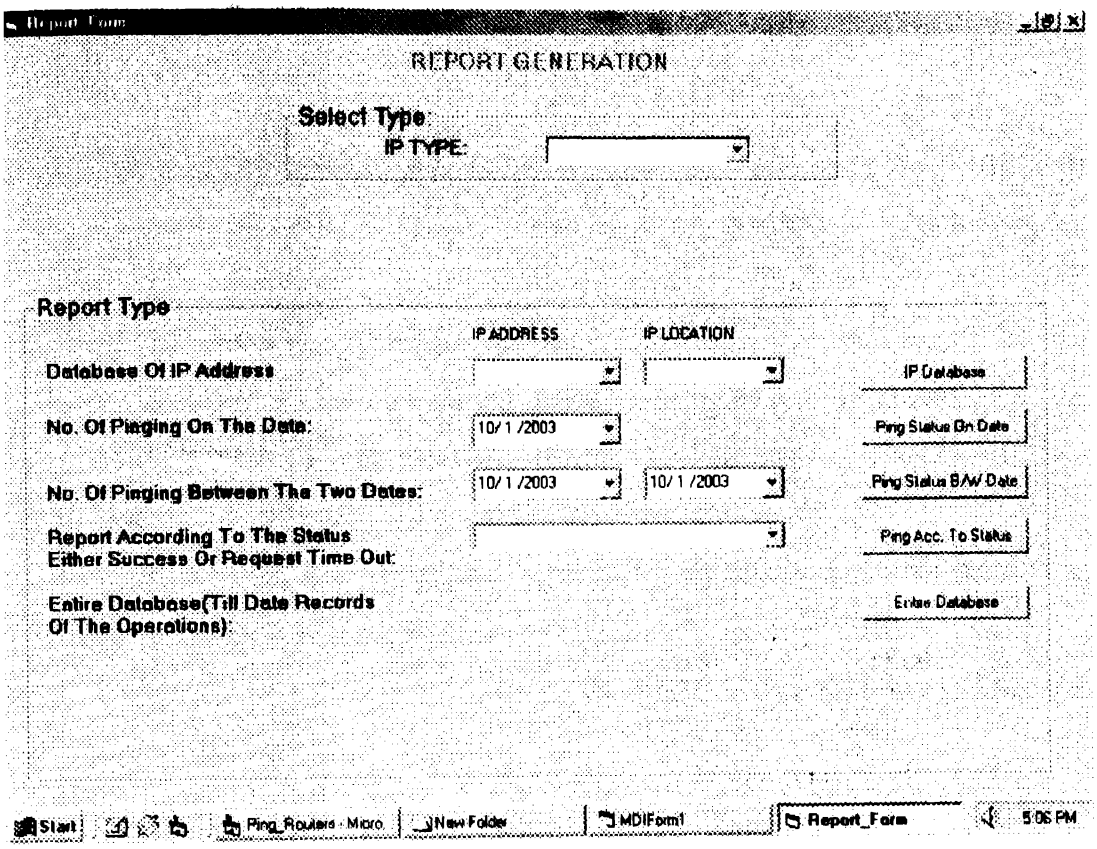
**4. GUI ping:** this option is used to ping the known address of the organigation . Its status is red if it cannot be pinged ,and green if it is able to ping. it read its data( the names and IP address ) from the ini files. the main file is abc.ini.




**5. Ping any:** this option can be used to ping any IP address and it return the status ,round trip time and data packet size.


Pinging Single Router
Ping..
IP Address -
PING
EXIT
Results..
Return Status
Round Trip Time -
Data Packet Size -

5. **Report**: this form displays the reports of previous pings



# FEATURES

1. The main purpose of this software is to view and ping the various IP addresses of the organization. we can ping type of network technologies.
   *(i)* *LAN*
   *(ii)* *WAN*
2. this program is also able to add, delete and update new IP addressed to the existing database. The addressed are added on the basis of LAN and WAN.
3. Easy to create reports using data environment of visual basic.
4. Every module, sub-procedure, function is fully testified, so that it make an accurate result without any controversy.
5. User-friendly interface that can easily interact with a common user and remove all the complications.
6. It displays a tool tip when a mouse is moved over buttons. The tool tip shows a comment, related to the description area of that button.
7. To achieve high performance, flexibility and better interface we have used ***Visual Basic 6.0.***

# SOFTWARE TESTING

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding.

**Type of Testing**

In this software project three types of testing were performed:

1. Unit testing.
2. Integration testing.
3. Validation testing.

## 1. UNIT TESTING

Unit testing focuses on verification efforts on the smallest unit of software design i.e. the module. It is always white-box oriented and the steps can be conducted in parallel for multiple modules. Loop testing was performed for each loop to check the higher and the lower boundary values for the loop and finds the loop termination condition and what type of condition becomes false and where the control is shifted when the loop terminates in-between execution. Boundary testing was done to check the actions that took place when user supplied the input for lowest and highest values.

## 2. INTEGRATION TESTING

After all the modules were connected with each other, integration testing was done to check all the modules are properly connected or not.

Regression testing was performed to check that if we modify any module then how the output is changed and what type of side effects are found on other modules and sub-modules. All the independent modules have been tested to check whether they produce the correct results or not.

## 3. VALIDATION TESTING

The system was tested repeatedly with a large number of different test cases and was found to functioning correctly.

**Type of Testing Windows**

There are four main windows that have been used while testing the application:

- The *Immediate window* to change the values of variables so that you can immediately see the effect of your changes. When the application is in break mode, you can evaluate expressions by typing in the Immediate window.
- The *Watch window* shows the current *watch expressions*, which are expressions, whose values you decide to monitor as the code runs. The Watch window can display a value for a watch expression only if the current statement is in the specified context. Otherwise, the Value column shows a message indicating the statement is not in context.
- The *Locals window* shows the value of any variables within the scope of the current procedure. As the execution switches from

procedure to procedure, the contents of the Locals window change to reflect only the variables applicable to the current procedure.

The *Output window* displays status messages at run time. With some language engines, this window is also used to display debugging strings generated in code.

# SOFTWARE MAINTENANCE

We define maintenance by describing four activities that can be undertaken after a program is released for use.

*The first* maintenance activity occurs because it is unreasonable to assume that software testing will uncover all latent errors in a large software system. During the use of any large program, errors may occur. The process that includes the diagnosis and correction of one or more errors is called corrective maintenance.

*The second* activity that contributes to a definition of maintenance because of the rapid change that is encountered in every aspect of computing. Adaptive maintenance is an activity that modifies software, to properly interface with a changing environment, is both necessary and commonplace.

*The third* activity that may be applied to a definition of maintenance occurs when a software package is successful. As the software is used, recommendations for new capabilities, modifications to exiting functions, and general enhancements are

received form users. To satisfy requests in this category, perfect maintenance is performed.

*The fourth* maintenance activity occurs when software is changed to improve future maintainability of reliability, or to provide a better basis for future enhancements. Often called preventive maintenance, this activity characterized by reverse engineering and reengineering techniques.

# GLOSSARY

Some seemingly complex technical terms commonly used in this application are defined/interpreted here under to enable the users to make comprehensible use of this system application.

## TECHNICAL TERMS:

**OCX Files:** A reusable, stand-alone software component often exposing a discrete subset of the total functionality of a product or application. An arbitrary number of ActiveX controls can be used as prefabricated components to aid in building a new application. ActiveX controls cannot run alone and must be loaded into a control container such as Microsoft Visual Basic or the Microsoft Internet Explorer. Formerly referred to as OLE control or OCX.

**DLL Files:** A .DLL file that contains one or more functions compiled, linked, and stored separately from the processes that use them. The operating system maps the DLLs into the process's address space when the process is starting up or while it is running. The process then executes functions in the DLL.

**EXE Files:** An EXE is an Active-X component that is compiled as an executable. The functionality of the Active-X EXE can be accessed by creating objects form a class contained within the EXE. In an addition, an ActiveX EXE can support stand-alone mode and be launched as a standard executable. An ActiveX EXE component is an implementation of out-process communication between a component and a client application.

**Active-X Controls**: A reusable software component that can add specialized functionality to World Wide Web sites, desktop application, development tools, and interactive video programs. These controls typically provide user interface elements, such as buttons. ActiveX controls were previously known as OLE custom controls or OCXes.

**Active-X Components**: An application or development tool that can use objects supplied by another application, or which exposes its own objects for use by another application. Formerly these were known as "OLE Automation servers" and "OLE Automation controllers."

**Properties**: A named attribute of an object. Properties define object characteristics, such as size and name, or the state of an object, such as enabled or disabled. Properties can be set for tables, columns, indexes, constraints, keys, and relationships.

**Methods**: A method in Visual Basic is a predefined action that operates on a specific object. It is just a single word that tells the program to do something to an object, such as form or control. There are a lot of methods in VB, like:

| Methods | Objectives |
|---|---|
| Drag Method<br>Method | Begins,ends orcancels a drag operation of a control Except the line, Menu , Shape, Timer, and Common Dialog controls.Does not support named arguments. |
| Move<br>Method | Moves a form of control.Doesn't support named Arguments. |
| Refresh<br>Method | Forces a complete repaint of a form or control. |
| Set Focus<br>Method | Moves the focus to the specified control or form. |

**Events**: Visual Basic is an event-driven-programming language. By event driven programming we mean defining what should happen (execute) when specific events occur. Examples of events are: clicking, double-clicking the mouse or pressing any key on the keyboard.

**Validation**: The process of checking whether entered data meets certain conditions or limitations. Mainly it is of two types:

1. Record-level rule: -- A validation rule, bound to a record that is activated when a record is inserted or changed, and most often used to verify data entry and correctness. Validation rules do not apply when records are deleted.

3. Field-level rule: -- A validation rule associated with a field that is activated when the field value is inserted or changed, and most often used to verify data entry and correctness.

**Interface**: A group of logically related operations or methods that provides access to a component object.

**Variables**: A location in the computer's memory where data is stored. You can change the contents of a variable but its name and storage area are reserved for use until you end the Visual Basic session or release the variable.

**Variable Scope**: Variable should always be defined with the smallest scope possible. VB can have the following scope.

| Scope | Where Variable Is Declared | Visibility |
|---|---|---|
| Procedure level | Event, Function, or Sub procedure | Visible in the procedure in Which it is declared |
| Script level | HEAD section of an HTML page, outside any procedure | Visible in every procedure In the script. |

**Procedure**: Procedures are building blocks of large programs. A procedure is a named sequence of statements executed as a unit and it instructs the application how to perform a certain task. Code within a module is organized in Procedures.

**Sub Procedure**: A Sub Procedure is a block of code that is executed in response to an event. By breaking the code in a module into Sub procedures, it becomes much easier to find or modify the code in your application.

**Function Procedures**: A function procedure executes the task and reports back to eh procedure from where it is called, returning a value. In other words, a function stores a value in a particular variable and when it ends, value stored is automatically returned to the calling procedure. Functions can return either Boolean, Integer, Long, Single, Double, Currency, Date, String, Variant, or Object data. By default it is of variant type.

**MDI**: The multiple-document interface (MDI) allows you to create an application that maintains multiple forms within a single container form. Applications such as Microsoft Excel and Microsoft Word for Windows have multiple-document interfaces.

**SDI**: The single-document interface (SDI) allows you to create an application that can maintain only a single form; an example of the SDI interface is the WordPad application included with Microsoft Windows (Figure 6.1). In WordPad, only a single document may be open; you must close one document in order to open another.

**Modules**: Code in Visual Basic is stored in modules. There are three kinds of modules: form, standard, and class. Simple applications can consist of just a single form, and all of the code in the application resides in that form module. As your applications get larger and more sophisticated, you add additional forms. Eventually you might find that there is common code you want to execute in

several forms. You don't want to duplicate the code in both forms, so you create a separate module containing a procedure that implements the common code.

```
rs1.Open "select count(*)  from Tbl_Details", con
rows_count = rs1(0)
rs1.Close
Results.Grid1.Clear

Results.Grid1.Row = 0
Results.Grid1.Col = 0
Results.Grid1.Text = "Serial No"
Results.Grid1.Col = 1
Results.Grid1.Text = "IP_Address"
Results.Grid1.Col = 2
Results.Grid1.Text = "Status"
Results.Grid1.Col = 3
Results.Grid1.Text = "TripTime"
Results.Grid1.Col = 4
Results.Grid1.Text = "DataSize"
Results.Grid1.Col = 5
Results.Grid1.Text = "Date"
Results.Grid1.Col = 6
Results.Grid1.Text = "Time"
Results.Grid1.Col = 7
Results.Grid1.Text = "OptionState"

Results.Grid1.Cols = 8
Results.Grid1.Rows = 2
col_count = Results.Grid1.Cols - 1

Results.Grid1.Row = Results.Grid1.Rows - 1
```

```
rs.Open "select * from Tbl_Details order by SerNo", con
'rs.MoveFirst

While Not rs.EOF
  For i = 1 To rows_count
    For j = 0 To col_count
      Results.Grid1.Col = j
      Results.Grid1.ColAlignment(j) = 1
      If rs.Fields(j) <> "" Then
        Results.Grid1.Text = rs.Fields(j)
      Else
        Results.Grid1.Text = ""
      End If

      If j < col_count Then Results.Grid1.Col = j + 1
    Next

    rs.MoveNext
    Results.Grid1.Rows = Results.Grid1.Rows + 1
    Results.Grid1.Row = Results.Grid1.Rows - 1
  Next
Wend

Results.Grid1.Rows = Results.Grid1.Rows - 1
rs.Close

con.Close
```

```
End Function


Public Function IP_Database()

Dim rows_count As Integer
Dim i As Integer, j As Integer
appath = App.Path
IP = Multi_Ping.txt_IP_View.Text
con.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & appath &
"\DB_Ping.mdb;Persist Security Info=False"
rs1.Open "select count(*) from Tbl_Details where IP_Address = '" & IP & "'", con
rows_count = rs1(0)
rs1.Close

If rows_count = 0 Then
   MsgBox "SORRY, NO RECORDS EXIST!", vbInformation
Else
   Results.Grid1.Clear

   Results.Grid1.Row = 0
   Results.Grid1.Col = 0
   Results.Grid1.Text = "Serial No"
   Results.Grid1.Col = 1
   Results.Grid1.Text = "IP_Address"
   Results.Grid1.Col = 2
   Results.Grid1.Text = "Status"
```

```
Results.Grid1.Col = 3
Results.Grid1.Text = "TripTime"
Results.Grid1.Col = 4
Results.Grid1.Text = "DataSize"
Results.Grid1.Col = 5
Results.Grid1.Text = "Date"
Results.Grid1.Col = 6
Results.Grid1.Text = "Time"
Results.Grid1.Col = 7
Results.Grid1.Text = "OptionState"

Results.Grid1.Cols = 8
Results.Grid1.Rows = 2
col_count = Results.Grid1.Cols - 1

Results.Grid1.Row = Results.Grid1.Rows - 1
rs.Open "select * from Tbl_Details where IP_Address = '" & IP & "' order by
SerNo ", con, adOpenDynamic
rs.MoveFirst

While Not rs.EOF
  For i = 1 To rows_count
    For j = 0 To col_count
      Results.Grid1.Col = j
      Results.Grid1.ColAlignment(j) = 1
      If rs.Fields(j) <> "" Then
        Results.Grid1.Text = rs.Fields(j)
      Else
```

```
            Results.Grid1.Text = ""
          End If

          If j < col_count Then Results.Grid1.Col = j + 1
        Next

        rs.MoveNext
        Results.Grid1.Rows = Results.Grid1.Rows + 1
        Results.Grid1.Row = Results.Grid1.Rows - 1
      Next
    Wend

    Results.Grid1.Rows = Results.Grid1.Rows - 1
    rs.Close

End If

con.Close

End Function
```

**2.GetHost_Name.bas**

```
Option Explicit
Private Const WSADescription_Len As Long = 256
Private Const WSASYS_Status_Len As Long = 128
```

```
Private Const WS_VERSION_REQD As Long = &H101
Private Const IP_SUCCESS As Long = 0
Private Const SOCKET_ERROR As Long = -1
Private Const AF_INET As Long = 2

Private Type WSADATA
  wVersion As Integer
  wHighVersion As Integer
  szDescription(0 To WSADescription_Len) As Byte
  szSystemStatus(0 To WSASYS_Status_Len) As Byte
  imaxsockets As Integer
  imaxudp As Integer
  lpszvenderinfo As Long
End Type

Private Declare Function WSAStartup Lib "wsock32" _
  (ByVal VersionReq As Long, _
   WSADataReturn As WSADATA) As Long

Private Declare Function WSACleanup Lib "wsock32" () As Long

Private Declare Function inet_addr Lib "wsock32" _
  (ByVal s As String) As Long

Private Declare Function gethostbyaddr Lib "wsock32" _
  (haddr As Long, _
   ByVal hnlen As Long, _
   ByVal addrtype As Long) As Long
```

```
Private Declare Sub CopyMemory Lib "kernel32" _
  Alias "RtlMoveMemory" _
 (xDest As Any, _
  xSource As Any, _
  ByVal nbytes As Long)

Private Declare Function lstrlen Lib "kernel32" _
  Alias "lstrlenA" _
 (lpString As Any) As Long

Public Function SocketsInitialize() As Boolean
Dim WSAD As WSADATA
  SocketsInitialize = WSAStartup(WS_VERSION_REQD, WSAD) =
  IP_SUCCESS
   End Function

Public Sub SocketsCleanup()
    If WSACleanup() <> 0 Then
      MsgBox "Windows Sockets error occurred in Cleanup.", vbExclamation
  End If
 End Sub

Public Function GetHostNameFromIP(ByVal sAddress As String) As String
  Dim ptrHosent As Long
  Dim hAddress As Long
  Dim nbytes As Long
    If SocketsInitialize() Then
```

```
        'convert string address to long
            hAddress = inet_addr(sAddress)
        If hAddress <> SOCKET_ERROR Then
         'obtain a pointer to the HOSTENT structure
      'that contains the name and address
      'corresponding to the given network address.
       ptrHosent = gethostbyaddr(hAddress, 4, AF_INET)
        If ptrHosent <> 0 Then
       'convert address and
        'get resolved hostname
         CopyMemory ptrHosent, ByVal ptrHosent, 4
         nbytes = lstrlen(ByVal ptrHosent)
      If nbytes <> 0 Then
           sAddress = Space$(nbytes)
           CopyMemory ByVal sAddress, ByVal ptrHosent, nbytes
           GetHostNameFromIP = sAddress
         End If

       Else: MsgBox "Call to gethostbyaddr failed."
       End If 'If ptrHosent
    SocketsCleanup
     Else: MsgBox "String passed is an invalid IP."
     End If 'If hAddress
   Else: MsgBox "Sockets failed to initialize."
  End If  'If SocketsInitialize
    End Function
'--end block--'
```

**3.IP_Detect.bas**

```
Option Explicit

'Description : This program helps to verify IP-level connectivity.
'             When troubleshooting, the ping command is used to
'             send an ICMP echo request to a target host name or
'             IP address. Use ping whenever you need to verify
'             that a host computer can connect to the TCP/IP network
'             and network resources. It verifies that a route exists
'             between the local computer and a network host by first
'             using ping and the IP address of the network host to
'             which you want to connect.
''

Private Const IP_SUCCESS As Long = 0
Private Const IP_STATUS_BASE As Long = 11000
Private Const IP_BUF_TOO_SMALL As Long = (11000 + 1)
Private Const IP_DEST_NET_UNREACHABLE As Long = (11000 + 2)
Private Const IP_DEST_HOST_UNREACHABLE As Long = (11000 + 3)
Private Const IP_DEST_PROT_UNREACHABLE As Long = (11000 + 4)
Private Const IP_DEST_PORT_UNREACHABLE As Long = (11000 + 5)
Private Const IP_NO_RESOURCES As Long = (11000 + 6)
Private Const IP_BAD_OPTION As Long = (11000 + 7)
Private Const IP_HW_ERROR As Long = (11000 + 8)
Private Const IP_PACKET_TOO_BIG As Long = (11000 + 9)
Private Const IP_REQ_TIMED_OUT As Long = (11000 + 10)
Private Const IP_BAD_REQ As Long = (11000 + 11)
Private Const IP_BAD_ROUTE As Long = (11000 + 12)
```

```
Private Const IP_TTL_EXPIRED_TRANSIT As Long = (11000 + 13)
Private Const IP_TTL_EXPIRED_REASSEM As Long = (11000 + 14)
Private Const IP_PARAM_PROBLEM As Long = (11000 + 15)
Private Const IP_SOURCE_QUENCH As Long = (11000 + 16)
Private Const IP_OPTION_TOO_BIG As Long = (11000 + 17)
Private Const IP_BAD_DESTINATION As Long = (11000 + 18)
Private Const IP_ADDR_DELETED As Long = (11000 + 19)
Private Const IP_SPEC_MTU_CHANGE As Long = (11000 + 20)
Private Const IP_MTU_CHANGE As Long = (11000 + 21)
Private Const IP_UNLOAD As Long = (11000 + 22)
Private Const IP_ADDR_ADDED As Long = (11000 + 23)
Private Const IP_GENERAL_FAILURE As Long = (11000 + 50)
Private Const MAX_IP_STATUS As Long = (11000 + 50)
Private Const IP_PENDING As Long = (11000 + 255)
Private Const PING_TIMEOUT As Long = 500
Private Const WS_VERSION_REQD As Long = &H101
Private Const MIN_SOCKETS_REQD As Long = 1
Private Const SOCKET_ERROR As Long = -1
Private Const INADDR_NONE As Long = &HFFFFFFFF
Private Const MAX_WSADescription As Long = 256
Private Const MAX_WSASYSStatus As Long = 128

Private Type ICMP_OPTIONS
  ttl           As Byte
  Tos           As Byte
  Flags         As Byte
  OptionsSize   As Byte
  OptionsData   As Long
```

```
End Type

Public Type ICMP_ECHO_REPLY
   Address        As Long
   Status         As Long
   RoundTripTime  As Long
   DataSize       As Long 'formerly integer
  'Reserved       As Integer
   DataPointer    As Long
   Options        As ICMP_OPTIONS
   data           As String * 250
End Type

Private Type WSADATA
   wVersion As Integer
   wHighVersion As Integer
   szDescription(0 To MAX_WSADescription) As Byte
   szSystemStatus(0 To MAX_WSASYSStatus) As Byte
   wMaxSockets As Long
   wMaxUDPDG As Long
   dwVendorInfo As Long
End Type

Private Declare Function IcmpCreateFile Lib "ICMP.DLL" () As Long

Private Declare Function IcmpCloseHandle Lib "ICMP.DLL" _
   (ByVal IcmpHandle As Long) As Long
```

```
Private Declare Function IcmpSendEcho Lib "ICMP.DLL" _
  (ByVal IcmpHandle As Long, _
   ByVal DestinationAddress As Long, _
   ByVal RequestData As String, _
   ByVal RequestSize As Long, _
   ByVal RequestOptions As Long, _
   ReplyBuffer As ICMP_ECHO_REPLY, _
   ByVal ReplySize As Long, _
   ByVal TimeOut As Long) As Long

Private Declare Function WSAGetLastError Lib "wsock32" () As Long

Private Declare Function WSAStartup Lib "wsock32" _
  (ByVal wVersionRequired As Long, _
   lpWSAdata As WSADATA) As Long

Private Declare Function WSACleanup Lib "wsock32" () As Long

Private Declare Function GetHostName Lib "wsock32" _
  Alias "gethostname" (ByVal szHost As _
   String, ByVal dwHostLen As Long) As Long

Private Declare Function GetHostByName Lib "wsock32" _
  Alias "gethostbyname" (ByVal szHost As String) As Long

Private Declare Sub CopyMemory Lib "kernel32" _
  Alias "RtlMoveMemory" _
 (xDest As Any, _
```

```
  xSource As Any, _
  ByVal nbytes As Long)

Private Declare Function inet_addr Lib "wsock32" _
  (ByVal s As String) As Long


Public Function GetStatusCode(Status As Long) As long
Dim msg As String
  Select Case Status
    Case IP_SUCCESS:          msg = "ip success"
    Case INADDR_NONE:         msg = "inet_addr: bad IP format"
    Case IP_BUF_TOO_SMALL:    msg = "ip buf too_small"
    Case IP_DEST_NET_UNREACHABLE:  msg = "ip dest net unreachable"
    Case IP_DEST_HOST_UNREACHABLE: msg = "ip dest host unreachable"
    Case IP_DEST_PROT_UNREACHABLE: msg = "ip dest prot unreachable"
    Case IP_DEST_PORT_UNREACHABLE: msg = "ip dest port unreachable"
    Case IP_NO_RESOURCES:     msg = "ip no resources"
    Case IP_BAD_OPTION:       msg = "ip bad option"
    Case IP_HW_ERROR:         msg = "ip hw_error"
    Case IP_PACKET_TOO_BIG:   msg = "ip packet too_big"
    Case IP_REQ_TIMED_OUT:    msg = "ip req timed out"
    Case IP_BAD_REQ:          msg = "ip bad req"
    Case IP_BAD_ROUTE:        msg = "ip bad route"
    Case IP_TTL_EXPIRED_TRANSIT:  msg = "ip ttl expired transit"
    Case IP_TTL_EXPIRED_REASSEM:  msg = "ip ttl expired reassem"
    Case IP_PARAM_PROBLEM:    msg = "ip param_problem"
    Case IP_SOURCE_QUENCH:    msg = "ip source quench"
```

```
    Case IP_OPTION_TOO_BIG:       msg = "ip option too_big"
    Case IP_BAD_DESTINATION:      msg = "ip bad destination"
    Case IP_ADDR_DELETED:         msg = "ip addr deleted"
    Case IP_SPEC_MTU_CHANGE:      msg = "ip spec mtu change"
    Case IP_MTU_CHANGE:           msg = "ip mtu_change"
    Case IP_UNLOAD:               msg = "ip unload"
    Case IP_ADDR_ADDED:           msg = "ip addr added"
    Case IP_GENERAL_FAILURE:      msg = "ip general failure"
    Case IP_PENDING:              msg = "ip pending"
    Case PING_TIMEOUT:            msg = "ping timeout"
    Case Else:                    msg = "unknown  msg returned"
  End Select

  GetStatusCode = CStr(Status) & "   [ " & msg & " ]"
  End Function
Public Function ping(sAddress As String, _
            sDataToSend As String, _
            ECHO As ICMP_ECHO_REPLY) As Long

  'If Ping succeeds :
  '.RoundTripTime = time in ms for the ping to complete,
  '.Data is the data returned (NULL terminated)
  '.Address is the Ip address that actually replied
  '.DataSize is the size of the string in .Data
  '.Status will be 0
  '
  'If Ping fails .Status will be the error code
```

```
Dim hPort As Long
Dim dwAddress As Long

'convert the address into a long representation
 dwAddress = inet_addr(sAddress)

'if a valid address..
 If dwAddress <> INADDR_NONE Then

  'open a port
   hPort = IcmpCreateFile()

  'and if successful,
   If hPort Then

    'ping it.
     Call IcmpSendEcho(hPort, _
                dwAddress, _
                sDataToSend, _
                Len(sDataToSend), _
                0, _
                ECHO, _
                Len(ECHO), _
                PING_TIMEOUT)

    'return the status as ping succes and close
     ping = ECHO.Status
     Call IcmpCloseHandle(hPort)
```

```
    End If

  Else:
     'the address format was probably invalid
     ping = INADDR_NONE
  End If
End Function

Public Sub SocketsCleanup()
  If WSACleanup() <> 0 Then
     MsgBox "Windows Sockets error occurred in Cleanup.", vbExclamation
  End If
End Sub

Public Function SocketsInitialize() As Boolean

  Dim WSAD As WSADATA
  SocketsInitialize   =   WSAStartup(WS_VERSION_REQD,   WSAD)
IP_SUCCESS

End Function
'--end block--'
```

## 4. clsTrace.bas

```
Option Explicit
Const SOCKET_ERROR = 0
Const IP_OPT_TS = &H44
Const IP_OPT_RR = &H7
Const AF_INET = 2
Private Type WSADATA
   wVersion As Integer
   wHighVersion As Integer
   szDescription(0 To 255) As Byte
   szSystemStatus(0 To 128) As Byte
   imaxsockets As Integer
   iMaxUdpDg As Integer
   lpVendorInfo As Long
End Type

Private Type Hostent
   h_name As Long
   h_aliases As Long
   h_addrtype As Integer
   h_length As Integer
   h_addr_list As Long
End Type

Private Type IP_OPTION_INFORMATION
   ttl As Byte
   Tos As Byte
```

```
    Flags As Byte
    OptionsSize As Long
    OptionsData As Long
End Type

Private Type IP_ECHO_REPLY
    Address  As Long
    Status As Long
    RoundTripTime As Long
    DataSize As Integer
    Reserved As Integer
    data As Long
    Options As IP_OPTION_INFORMATION
    ReturnedData    As String * 256
End Type

Private Declare Function GetHostByName Lib "wsock32.dll" Alias
"gethostbyname" (ByVal HostName As String) As Long
Private Declare Function WSAStartup Lib "wsock32.dll" (ByVal
wVersionRequired&, lpWSAdata As WSADATA) As Long
Private Declare Function WSACleanup Lib "wsock32.dll" () As Long
Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory"
(hpvDest As Any, hpvSource As Any, ByVal cbCopy As Long)
Private Declare Function IcmpCreateFile Lib "ICMP.DLL" () As Long
Private Declare Function IcmpCloseHandle Lib "ICMP.DLL" (ByVal HANDLE
As Long) As Boolean
Private Declare Function IcmpSendEcho Lib "ICMP" (ByVal IcmpHandle As
Long, ByVal DestAddress As Long, ByVal RequestData As String, ByVal
```

```
RequestSize As Integer, RequestOptns As IP_OPTION_INFORMATION,
ReplyBuffer As IP_ECHO_REPLY, ByVal ReplySize As Long, ByVal TimeOut
As Long) As Boolean
Private Declare Function gethostbyaddr Lib "wsock32.dll" (haddr As Long, ByVal
hnlen As Long, ByVal addrtype As Long) As Long
Private Declare Function lstrlenA Lib "kernel32" (ByVal Ptr As Any) As Long
Public Event UpdateRoute(lHop As Long, lTime As Long, sHostAddress As
String, sHostName As String, lAttempts As Long)
Public Event Error(sWhat As String)
Private m_CharsPerPacket As Long
Private m_TraceAttempts As Long
Private m_TraceHost As String
Private m_TraceHostName As String
Private m_ResolveHostname As Boolean
Private m_TraceHostIP As Long
Private m_Cancel As Boolean
Private hFile As Long
Public Property Get CharsPerPacket() As Long
   CharsPerPacket = m_CharsPerPacket
End Property
Public Property Let CharsPerPacket(lNewCharCount As Long)
   m_CharsPerPacket = lNewCharCount
   If m_CharsPerPacket < 32 Then m_CharsPerPacket = 32
   If m_CharsPerPacket > 128 Then m_CharsPerPacket = 128
End Property
Public Property Get TraceAttempts() As Long
   TraceAttempts = m_TraceAttempts
End Property
```

```
Public Property Let TraceAttempts(lNewTraceAttempts As Long)
   m_TraceAttempts = lNewTraceAttempts
   If m_TraceAttempts < 1 Then m_TraceAttempts = 1
   If m_TraceAttempts > 20 Then m_TraceAttempts = 20
End Property
Public Property Get TraceHost() As String
   TraceHost = m_TraceHost
End Property

Public Property Let TraceHost(sNewTraceHost As String)
   m_TraceHost = sNewTraceHost
   If m_TraceHost = "" Then m_TraceHost = "www.indianoil.co.in"
   m_TraceHostIP = GetIPAddress(m_TraceHost)
End Property

Public Property Get ResolveHostname() As Boolean
   ResolveHostname = m_ResolveHostname
End Property
Public Property Let ResolveHostname(bNewResolveHostname As Boolean)
   m_ResolveHostname = bNewResolveHostname
End Property
Public Property Get HostIP() As String
   HostIP = GetIPFromLong(m_TraceHostIP)
End Property
Private Function GetIPFromLong(lInput As Long) As String
   Dim bBytes(0 To 3) As Byte
   CopyMemory bBytes(0), lInput, 4
```

```
  GetIPFromLong = CStr(bBytes(0)) + "." + CStr(bBytes(1)) + "." +
CStr(bBytes(2)) + "." + CStr(bBytes(3))
End Function
Private Function GetIPAddress(HostName As String) As Long
  Dim hHostent As Hostent, AddrList As Long
  If GetHostByName(HostName + String(64 - Len(HostName), 0)) <>
SOCKET_ERROR Then
    CopyMemory hHostent.h_name, ByVal GetHostByName(HostName +
String(64 - Len(HostName), 0)), Len(hHostent)
    CopyMemory AddrList, ByVal hHostent.h_addr_list, 4
    CopyMemory GetIPAddress, ByVal AddrList, 4
  End If
End Function
Private Function GetHostName(ByVal Address As Long) As String
  Dim lLength As Long, lRet As Long
  lRet = gethostbyaddr(Address, 4, AF_INET)
  If lRet <> 0 Then
    CopyMemory lRet, ByVal lRet, 4
    lLength = lstrlenA(lRet)
    If lLength > 0 Then
      GetHostName = Space$(lLength)
      CopyMemory ByVal GetHostName, ByVal lRet, lLength
    End If
  Else
    GetHostName = ""
  End If
End Function
Private Sub Class_Initialize()
```

```
    Dim lpWSAdata As WSADATA
    Call WSAStartup(&H101, lpWSAdata)
    CharsPerPacket = 32
    TraceAttempts = 4
    'TraceHost = "www.indianoil.co.in"
    ResolveHostname = False
End Sub
Public Function StartTrace() As Boolean
    Dim TimeToLive As Long, lAttempts As Long
    Dim OptInfo As IP_OPTION_INFORMATION
    Dim EchoReply As IP_ECHO_REPLY
    m_Cancel = False
    hFile = IcmpCreateFile()
    If hFile = 0 Then
        RaiseEvent Error("Unable to create ICMP handle!")
        Exit Function
    End If
    lAttempts = 1
    For TimeToLive = 1 To 255
        OptInfo.ttl = TimeToLive
        If IcmpSendEcho(hFile, m_TraceHostIP, String(m_CharsPerPacket, "A"),
m_CharsPerPacket, OptInfo, EchoReply, Len(EchoReply) + 8, 100) = 1 Then
            If m_ResolveHostname Then
                RaiseEvent UpdateRoute(TimeToLive, EchoReply.RoundTripTime,
GetIPFromLong(EchoReply.Address), GetHostName(EchoReply.Address),
lAttempts)
            Else
```

```
            RaiseEvent UpdateRoute(TimeToLive, EchoReply.RoundTripTime,
GetIPFromLong(EchoReply.Address), "", lAttempts)
          End If
          lAttempts = 1
        Else
          If lAttempts < m_TraceAttempts Then
            TimeToLive = TimeToLive - 1
          Else
            RaiseEvent Error("Trace failed after " + CStr(lAttempts) + "
attempt(s).")
            Exit For
          End If
          lAttempts = lAttempts + 1
          EchoReply.Address = 0
        End If
        If m_TraceHostIP = EchoReply.Address Then
          StartTrace = True
          Exit For
        End If
        DoEvents
        If m_Cancel = True Then Exit For
      Next TimeToLive
      IcmpCloseHandle hFile
    End Function
    Public Sub CancelTrace()
      m_Cancel = True
      IcmpCloseHandle hFile
    End Sub
```

```
Private Sub Class_Terminate()
   CancelTrace
   Call WSACleanup
End Sub
```

## 5. MDIForm1.frm

```
Private Sub add_Click()
New_User_Form.Show
End Sub

Private Sub exit_Click()
End
End Sub

Private Sub gui_ping_Click()
NewOne.Show
End Sub

Private Sub mnureport_Click()
Report_Form.Show
End Sub

Private Sub multi_ping1_Click()
Multi_Ping.Show
End Sub
```

```
Private Sub new_user_Click()
New_User_Form.Show
End Sub
Private Sub trace_ip_Click()
Trace_IP.Show
End Sub
Private Sub single_Click()
Ping_IP.Show
End Sub

Private Sub trace_ip1_Click()
Trace_IP.Show
End Sub
```

# REFERENCES

The following books, websites, magazines, newspapers, existing software (and their help) have been referred to in the development of this software application:


- Mastering Visual Basic 6.0, Evangelos Petroutsos, BPB Publication
- Fundamental of Software Testing, Ghezzi. Et Al, Prentince-Hall of India
- Teach Yourself Visual Basic 6 in 21 Days, Perry, BPB Publication
- System Analysis and Design Second Edition, Elias M.Awad, Galgotia Publication.
- www.msdn.microsoft.com
- www.google.com
- www.whatis.com

- Study material of NIIT
- Study material of CMC
- Study material of ET&T